Introduction

Why memory subsystem design is important
- CPU speeds increase 55% per year
- memory speeds increase 7% per year
- rate of increase is also widening

Solutions
- eliminate memory operations
- decrease miss penalty
- decrease cache/memory access times
- hide memory latencies

Introduction

Memory hierarchies work
- principal of **locality of reference**
  - temporal
  - spatial
  - processor
  - value
- technology
  - speed vs. size

Cache Review

Organization

- Size
- Block
- Set
- Associativity
  - direct-mapped
  - set-associative
  - fully-associative

Block replacement algorithm

- LRU
- random

Metrics

Hit (miss) ratio

$$\frac{\#\text{hits(misses)}}{\#\text{references}}$$

Effective access time

$$\text{Time}_{\text{hit}} + \text{MissRatio} \bullet \text{Time}_{\text{miss}}$$

Misses per instruction

$$\frac{\text{misses}}{\text{instruction}} = \frac{\text{memory accesses}}{\text{instruction}} \bullet \text{miss rate}$$

Execution time including the memory system

$$\frac{\text{instructions}}{\text{program}} \bullet (\text{CPI}_{\text{CPU}} + \text{CPI}_{\text{memory}}) \bullet \text{cycle time}$$

$$\text{CPI}_{\text{memory}} = \frac{\text{memory accesses} \bullet \text{miss rate} \bullet \text{miss penalty}}{\text{instruction}}$$

Miss Classification

Compulsory

- increase block size

Capacity

- increase cache size

Conflict

- increase associativity
- victim cache

Coherence

- increase block size + improve processor locality

Design Tradeoffs

Cache size

- hit ratio (capacity misses)

  vs.

- access time

Design Tradeoffs

Block size

- hit ratio (compulsory/coherence misses)
- amortize overhead of block transfers
- tag overhead

  vs.

- pollution point
- false sharing

Design Tradeoffs

Associativity

- hit ratio (conflict misses)

  vs.

- tag size
- comparators (HW cost)
- MUX (time cost)
- block replacement hardware

Why does 2-way set associative cache have a lower miss ratio than a direct mapped cache of the same size?

Design Tradeoffs

Block replacement

- LRU
- random

When would LRU produce its worse behavior?

Design Tradeoffs

Block allocation on a write miss

- write-allocate
- no-write-allocate

Design Tradeoffs

Memory update policy

- **write through**
  - performance depends on the # of writes
  - write buffer
    - read miss check (A X)
    - store compression (A S)
  - match with no write allocate
- **write back**
  - performance depends on the # of block replacements but...
  - dirty bit
  - more complex implementation
  - tag check before the write

  - optimization: fetch before replace (A)

Design Tradeoffs

Cache contents

- **separate** instruction & data caches
  - separate access $\Longrightarrow$ double the bandwidth
  - different configurations for I & D: why?
  - shorter access time
- **unified** cache
  - lower miss rate but possibly lower effective access time
  - less cache controller hardware

Design Tradeoffs

Virtual or physical addressing

- hit time
  - no address translation (v) (S)
  - address translation in parallel with cache access (p) (A, P)
    - what are the restrictions?
    - how can you get around them?
- compiler support for data placement (v)
- context switch (v)
  - cache flushing
  - PIDs
- synonyms (v)
  - allocation restrictions (S